BBYTE Updating

Efficiency option

Sun, Jul 28, 2002

The BBYTE system table represents the digital data pool in an IRM or PowerPC-based Linac-style system. It resides in nonvolatile memory because it is the basis of the digital part of the Settings Restore operation that occurs during system initialization following a reset; therefore, the most recent values of digital data must be kept in this table. During the Update task execution at the beginning of every 15 Hz cycle, the digital data pool must be updated. An entry in the Data Access Table specifies that the BADDR Binary Address table be scanned and each address found therein accessed to obtain one byte that is to be placed in the BBYTE table. This note describes a means of limiting the number of accesses to nonvolatile memory needed to accomplish this step.

One way to improve efficiency is to use a copy of the BADDR table that is maintained in dynamic (fast) memory. Since the contents of this nonvolatile system table, when edited, is normally edited via the general purpose Memory Dump page application, the system code has no way to know when it has been altered. This implies that the system code needs to monitor the contents of the table occasionally to detect changes. And “occasionally”should mean often enough so that making a change results in reasonably prompt alteration in the updating of the digital data pool.

During the update of the digital data pool, let a copy of it be made first in volatile memory. After this copy has been created by accessing the BADDR addresses, copy from this copy into BBYTE 4 bytes at a time, to reduce the number of write accesses by a factor of 4. In addition, a fast memory copy of BBYTE can be maintained as well, and recognizing that most digital data does not often change, the copy into BBYTE can be limited to changed data.

Most entries in BADDR refer to nonvolatile addresses, so that every use of such entries costs a nonvolatile memory access. If these entries were modified to point to fast memory, this would not occur. During the Settings Restore operation following system reset, the contents of this fast memory would be updated to a copy of what is stored in the BBYTE table, which must always be in nonvolatile memory.

What are the changes being suggested here? As described in the preceding paragraph, change all the dummy addresses in the BADDR table to fast memory. Keep a copy of the BADDR table for routine review during Data Access table processing. Update this copy from time to time by copying from the actual nonvolatile BADDR table. When building the bytes of digital data, store such data at first into a scratch array, copying it into the fast copy of the BBYTE table only when finished. Compare the fast copy with another “carbon copy”of the BBYTE table, writing into the real BBYTE table only when changes are detected. This makes sure that the BBYTE table always has the up-to-the-current-cycle digital data.

The result of these changes is a great reduction in the number of accesses to nonvolatile memory attendant with updating the digital data pool each 15 Hz cycle. Suppose the very simple default case of 128 entries in the BADDR table all of which are the default set of now nonvolatile memory addresses. By changing these default addresses to fast memory, we save 128 accesses each cycle, compared to what is done now. By first writing the digital data into a copy of BBYTE that is actually the online digital data pool, we save another 128 accesses. By keeping a copy of the last data recording in the nonvolatile BBYTE table, there may be no writes to the slow BBYTE table, assuming that no changes have occurred in the digital data

readings.

What can these tables be called? The BBYTE table as recorded in the nonvolatile table directory is in the actual nonvolatile area of memory. The entry for BBYTE in the fast online table directory will point to the copy of what is currently in BBYTE. This copy will be initialized to the actual BBYTE table contents at reset time. But this means that every reference to update any of the BBYTE will only target the copy that will be checked against the real copy of BBYTE every cycle. Let us call BBYTE1 that online copy of BBYTE. Then BBYTE2 can be the fast copy of BBYTE that is always correct.This BBYTE2 would only be updated when BBYTE1 is compared against it, and every change that is found is copied both into BBYTE2 and into the slow BBYTE.

If a setting is made at some point in the cycle, it can have immediate influence, because BBYTE1 will be changed, and direct references to BBYTE actually point to BBYTE1 during system operation. But once each cycle, BBYTE1 will be compared against BBYTE2, and any changes will be written both to BBYTE2 and to the real BBYTE. The only time that BBYTE2 is written to is when the same data is also copied into BBYTE.

In the case that downloading of the entire BBYTE is being performed, and if BBYTE1 is actually targeted, then both BBYTE2 and BBYTE will be updated at the start of the very next 15 Hz cycle.

What if a BADDR table entry is updated via the listype (34) that targets those table entries? In this case, if the online table directory refers to the fast copy of BADDR, the setting will update only that entry. A fix for this is to use a special setting type routine for this listype that would target an entry in both the online and permanent copies of BADDR.

What if the entire BADDR table is downlaoded via memory access to the base address in the online table directory? This would target the online BADDR1, not the nonvolatile BADDR.

What code changes are needed to implement this scheme?

At reset time, create an online copy of both BADDR and BBYTE, maybe called BADDR1 and BBYTE1. Install pointers to these online copies in the online table directory, so that normal system operation references the online copies rather than the nonvolatile tables. In addition, also create a second fast memory copy of BBYTE called BBYTE2, which will always contain a copy of what is in the nonvolatile BBYTE table.

Implement a new setting type routine for support of listype 34 so that targeting BADDR entries via that listype will actually write into both the online BADDR1 and the nonvolatile BADDR entries.